Collaborative zkSNARKs

Zero-knowledge proofs for *distributed* secrets

Alex Ozdemir, Dan Boneh

To appear in USENIX Security'22

Provable properties about secrets

<u>Authentication</u> ☑ zkSNARKs

I prove that I know my password/secret key.

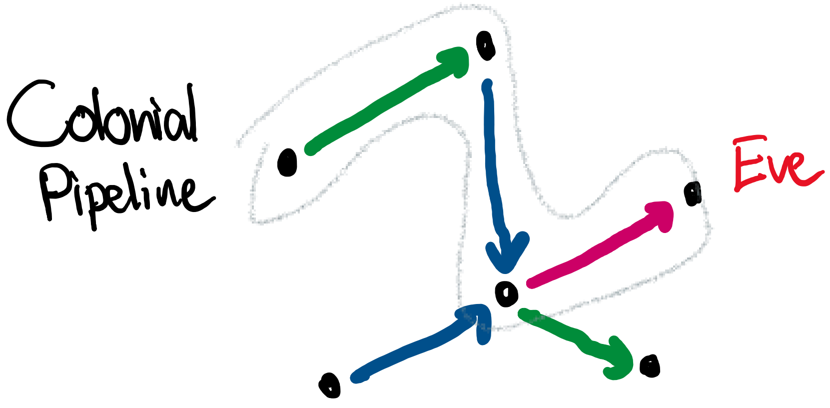
<u>Client(secret)</u>

<u>Server</u>

Provable properties about *distributed* secrets

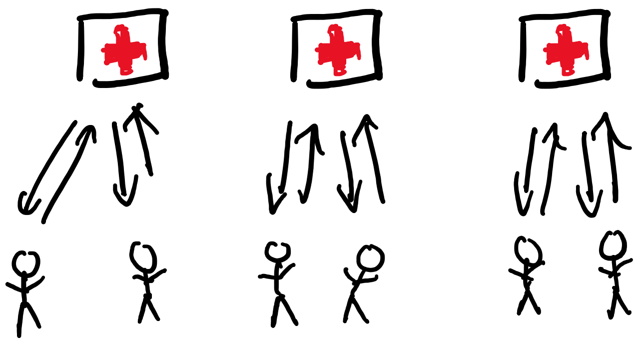
Money Laundering

Several banks prove that "Eve" has Colonial Pipeline's ransom.



Healthcare Statistics

Several hospitals prove that procedure prices are "fair".



This talk

Applications:

- Authentication ✓ zkSNARKs
- Money laundering ?
- Healthcare statistics ?

Outline:

1. Why zkSNARKs are insufficient
2. New tool: ***collaborative*** **zkSNARK**
3. Building collaborative zkSNARKs
4. Surprising efficiency

Background

zkSNARKs

Witness relations

$\exists ?\, \omega.\ (x,\omega) \in R$

$\underline{P(x,\omega)}$ $\qquad\qquad$ $\underline{V(x)}$

$$\xrightarrow{\ \omega\ } \quad (x,\omega) \overset{?}{\in} R$$

$$\downarrow$$

$$\{0,1\}$$

- w may be large
- not private

zkSNARKs

$pp \leftarrow \mathrm{Setup}()$

$\exists ?\, \omega.\ (x, \omega) \in R$

$P(x, \omega)$ $\qquad$ $V(x)$

$\pi \leftarrow \mathrm{Prove}(pp, x, \omega)$ $\xrightarrow{\ \pi\ }$ $\mathrm{Verify}(pp, x, \pi)$

$\downarrow$

$\{0,1\}$

- Sound: π proves w exists
- (zk) zero knowledge: hides w
- (S) succinct: short π, fast Verify
- (N) non-interactive
- (AR) argument: *computationally* sound
- (K) knowledge: P *knows* w

Authentication with a zkSNARK

pk

sk

Client → $P(x, \omega)$

Server → $V(x)$

$(x, \omega) \overset{?}{\in}$ Keypairs

$\pi \leftarrow \text{Prove}(pp, x, \omega)$

$\xrightarrow{\pi}$

"I know sk"

$\text{Verify}(pp, x, \pi)$

↓

aunthenticate?

$\{0, 1\}$

Existing zkSNARKs

- From pairings & elliptic curves
  - Groth16
  - Marlin (KZG)
  - Plonk (KZG)
  - …
- From hashing & codes
  - Fractal
  - …
- …

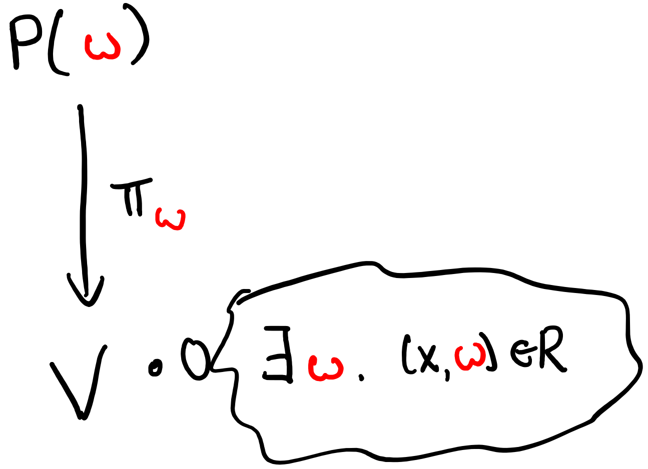
For *distributed* secret data, who plays the prover?

[G’16][GWC’19][KZG’10][CHMMVW’20][COS’20]…

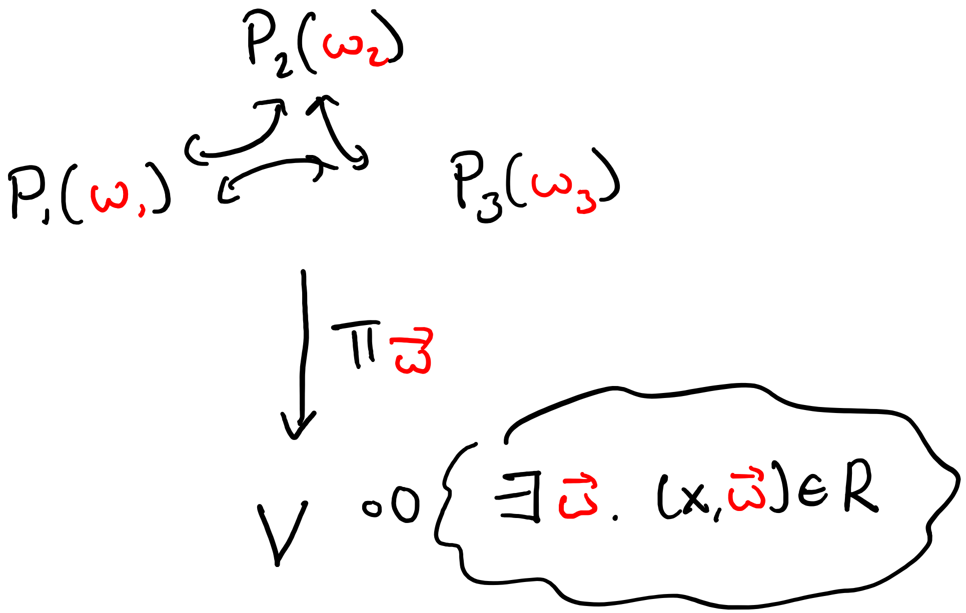
Collaborative zkSNARKs

Definitions

Collaborative zkSNARKs



conventional



collaborative

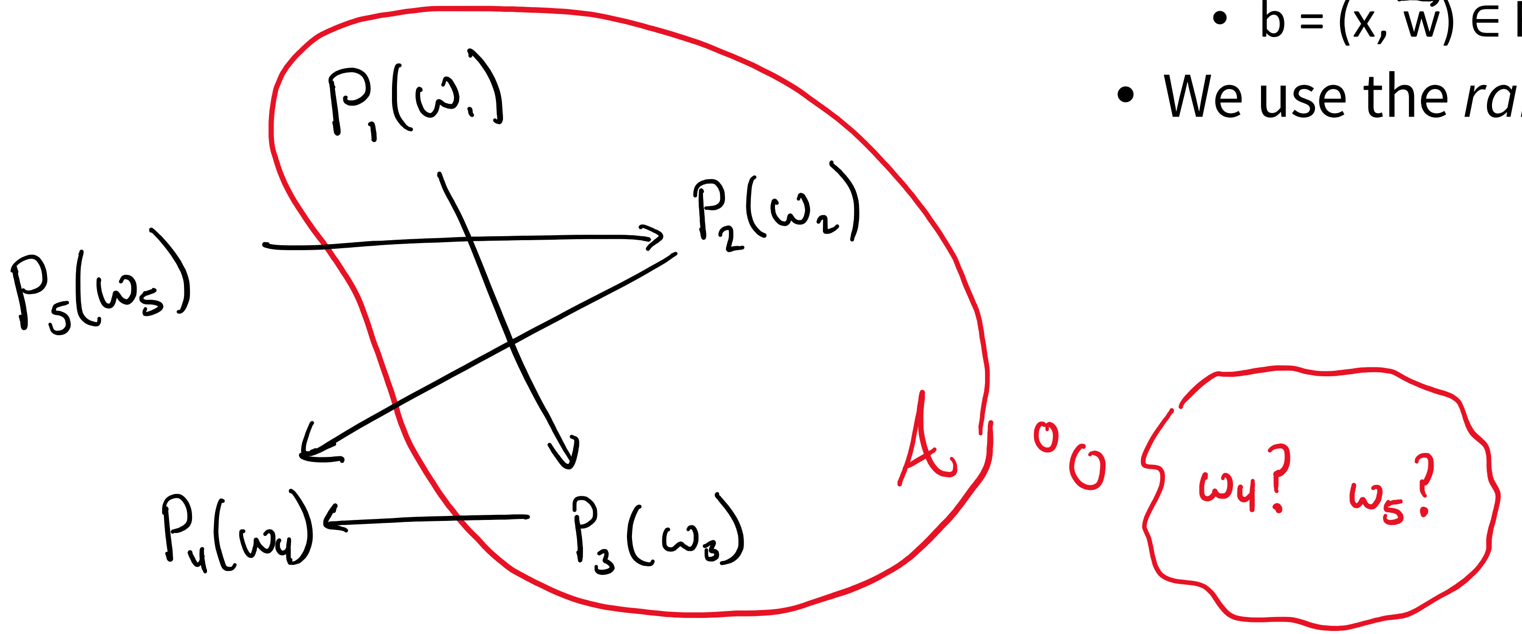
Syntax:

- $\text{Setup}() \rightarrow pp$
- $\boxed{\text{Prove}(pp, x, P_1(w_1), \ldots, P_N(w_N)) \rightarrow \pi}$
- $\text{Verify}(pp, x, \pi) \rightarrow \{0, 1\}$

t-zero-knowledge

Any adversary controlling ≤t provers learns nothing but whether

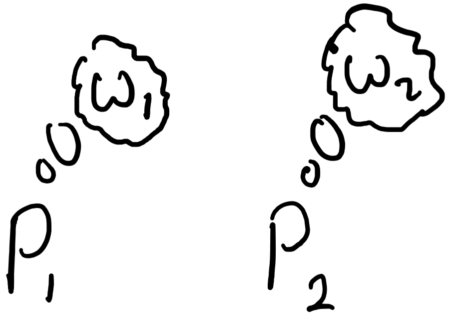
$(x, \vec{w}) \in R$

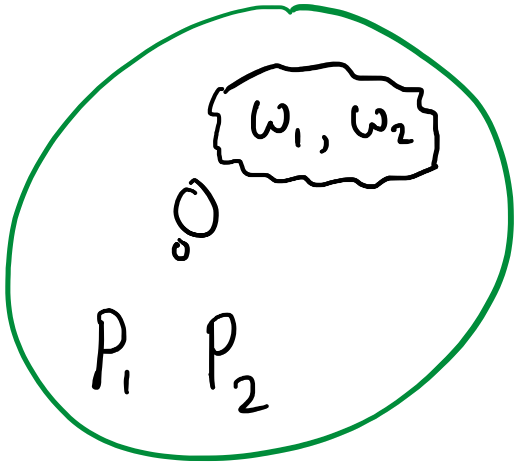


- Formally
  - Adversary corrupts ≤t provers
  - ZK simulator is given
    - the corrupt witnesses
    - x
    - $b = (x, \vec{w}) \in R$
  - We use the *random oracle model*

Knowledge soundness

- *could* mean P_1 knows w_1, …, P_N knows w_N
- actually means P_1, …, P_N collectively know w_1, …, w_N
  - “distributed knowledge” [Halpern, Moses ’90]
- Random oracle
  - extractor programs RO

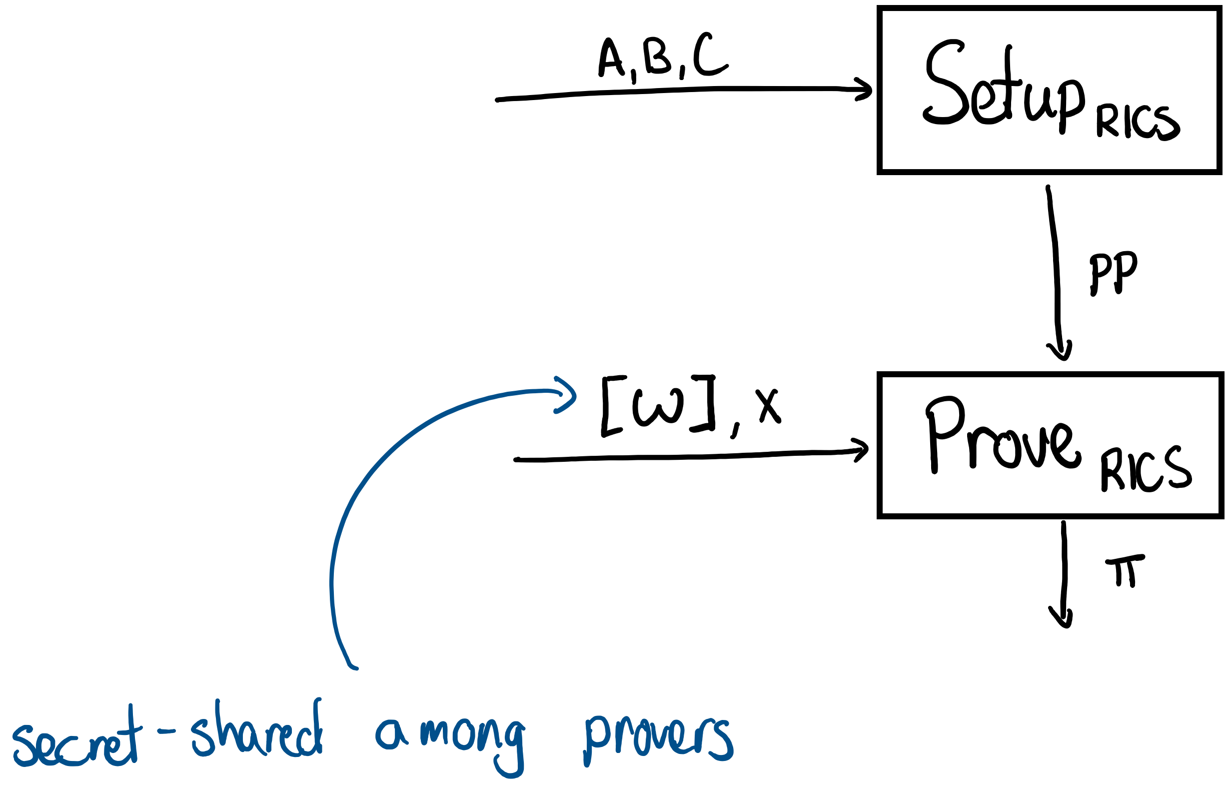




Our focus: secret-shared R1CS witnesses

R1CS:

- Class of relations
- Generalize arithmetic circuits
- Definition:
  - R: $A, B, C \in \mathbb{F}^{n \times m}$
  - $x \in \mathbb{F}^k, w \in \mathbb{F}^{m-k}$
  - Satisfied when
    - $a \leftarrow w \parallel x$
    - $Aa \circ Ba = Ca$

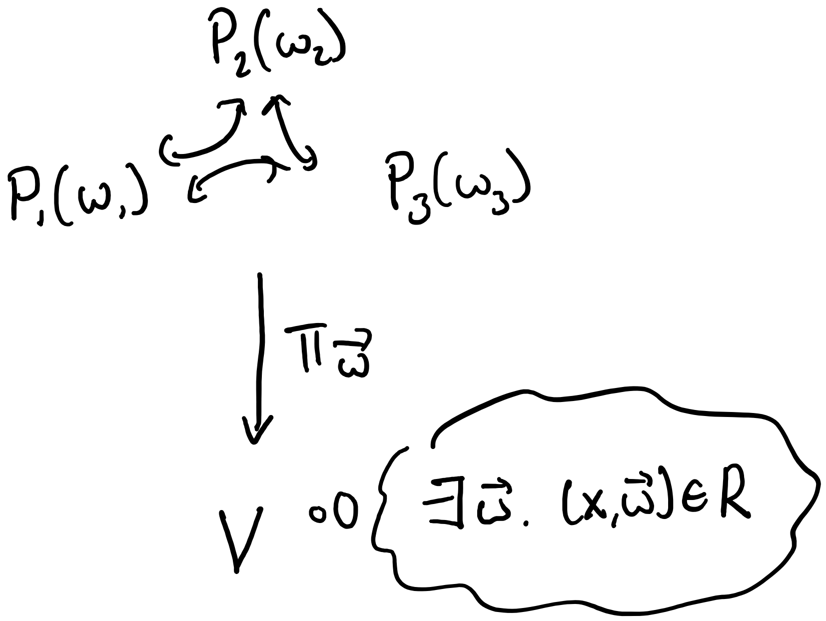


Designing co-zkSNARKs

Overview of constructions

Approach: MPC the Prover

$$\underbrace{\underbrace{\mathrm{GenericMPC}}_{1000\times\ \text{slower}}(\underbrace{\mathrm{zkSNARK.Prove}}_{1000\times\ \text{slower}}, [w])}_{1{,}000{,}000\times\ \text{slowdown ?!}} \to \pi$$



avoid. instead acheive 1000x - 2000x slowdown

Potential Bottlenecks

Single-prover bottlenecks:

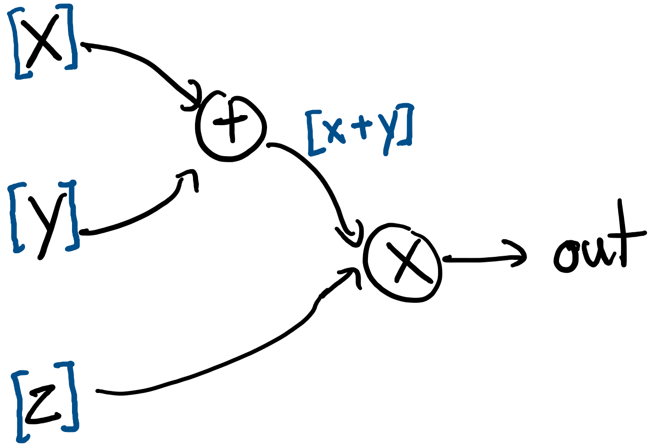
- Elliptic curve operations — This talk: a good solution
- Fourier transforms — MPC-efficient

MPC bottlenecks:

- Polynomial divisions — MPC-efficient (for SNARK provers)
- Partial products — special protocol
- Merkle tree evaluations — This talk: an okay solution

MPC Crash-Course

Computation: arithmetic circuit over a finite field



1. Secret-share wire values among N parties

   e.g. $x = \underbrace{x^{(1)} + x^{(2)} + \cdots + x^{(N)}}_{[x]}$

2. Secure protocols for +, * on shares

3. Evaluate circuit, inputs to outputs

We use two MPCs: SPDZ (authenticated additive shares, malicious majority) and GSZ (Shamir shares, honest majority)

MPC-friendly elliptic curve arithmetic

Elliptic curve addition

$g_1 = (x_1, y_1)$
$g_2 = (x_2, y_2)$ $\in E(\mathbb{F})$

$g_1 \boxplus g_2$?

Option 1: Share (x, y) coordinates

$[g_1] \boxplus [g_2] =$

$([x_1],[y_1]) \boxplus ([x_2],[y_2]) \longrightarrow$ $x = \frac{x_1y_2+x_2y_1}{1+dx_1x_2y_1y_2}$ $y = \frac{\ldots}{1-dx_1x_2y_1y_2}$

Nasty Formulas

$\rightarrow$ Expensive

- multiple multiplications
- communication

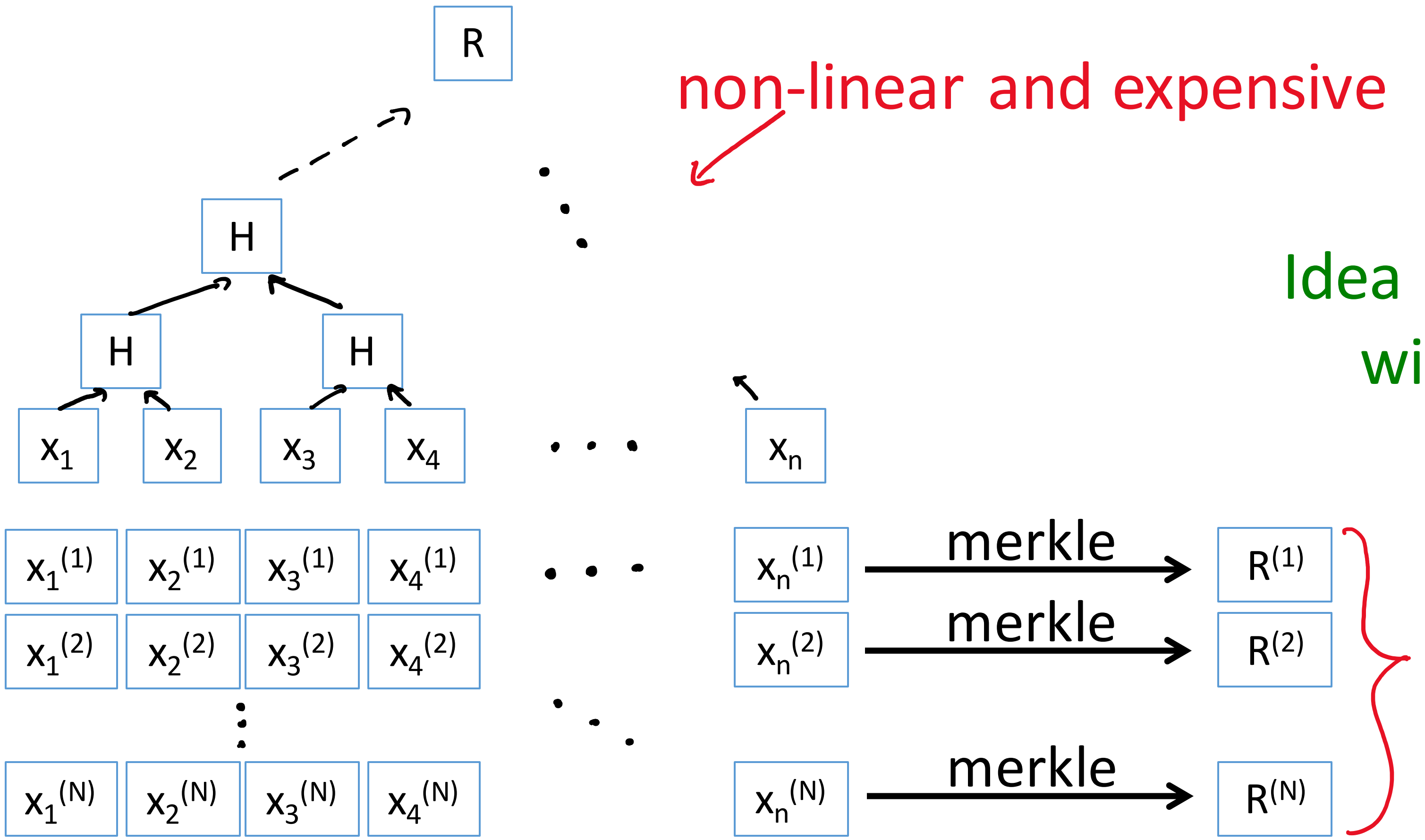
Option 2: Elliptic curve sharing

$[g] = g^{(1)} \boxplus g^{(2)} \boxplus \cdots \boxplus g^{(N)} \rightarrow$ add share-wise $\rightarrow$ Efficient

- no communication

Merkle Trees

non-linear and expensive



Idea (Riad S. Wahby): reduce with proof composition

proof size (& verifier work) $\times N$

Implementation

Implementation Goals

- Three base zkSNARKs
  - Groth16
  - Marlin/KZG
  - Plonk/KZG
- Two base MPCs
  - GSZ: $t < N/2$ (honest majority)
  - SPDZ: $t < N$ (malicious majority)

- Goals:
  - Compete with existing zkSNARKs
    - (well optimized!)
  - Iterate on MPCs, sub-protocols
  - Don't work too hard

An Opportunity

1. Arkworks has curve-generic provers:

```
fn prove<E: PairingEngine>(..) {
  …
}
```

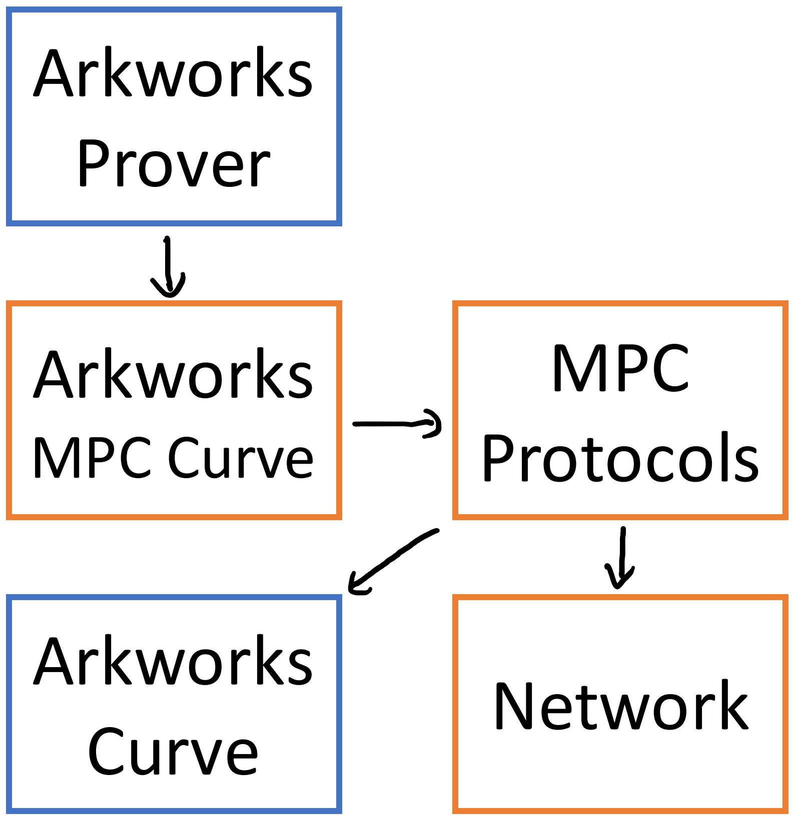
2. Curve interfaces define +, *, …

```
trait PairingEngine {
    type ScalarField;
    type Curve;
    fn field_add(…) …;
    fn field_mul(…) …;
    fn curve_add(…) …;
    …
}
```

*Radically oversimplified

Implementation Strategy

1. Implement MPCs for shared field and curve operations
    1. SDPZ
    2. GSZ
2. Wrap MPCs & implement arkworks interfaces
3. Instantiate zkSNARK prover
    - Mis-appropriates zkSNARK prover as a co-zkSNARK prover!



Performance

Experimental Setup

Measure:

- Wall-clock proving time

Vary:

- N: number of provers
- n: R1CS size (# constraints)
- c: link capacity
- Base: Groth16/Marlin/Plonk
- t: security threshold
  - $< N/2$ (honest majority, GSZ)
  - $< N$ (malicious majority, SPDZ)

Simplifications:

- No intra-prover parallelism
- Skip MPC preprocessing
  - Small for our computation

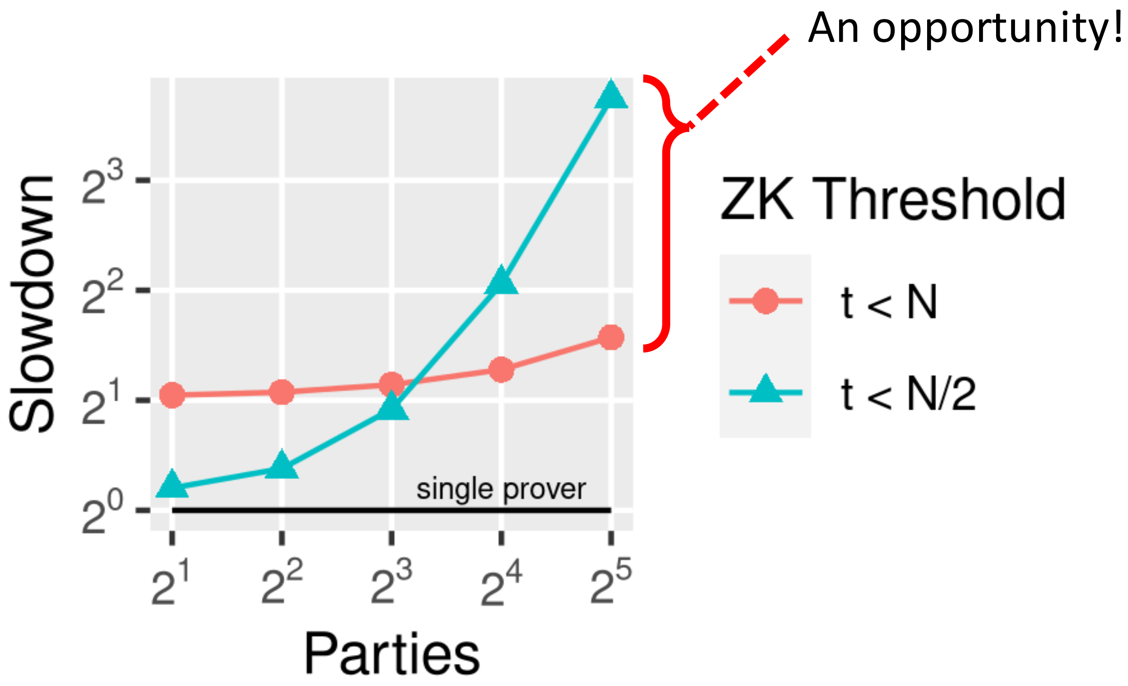
Experiment 1: Good network, few parties

Fix a 3Gb/s link, vary # rank-1 constraints

$t < N/2 \rightarrow$ no slowdown $\quad\quad$ $t < N \rightarrow$ 2x slowdown

Experiment 2: Many provers

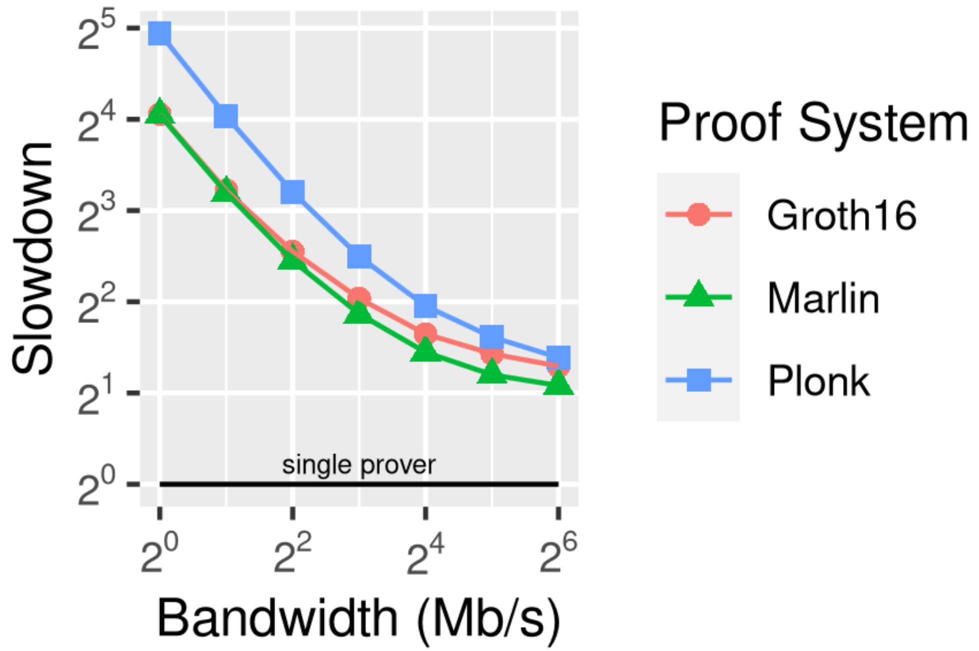
Fix 1024 constraints, 3Gb/s link, Groth16, vary # of provers



Slowdown grows with N; better for SPDZ

Experiment 3: Low-capacity link

Fix 1024 constraints, 2 provers, malicious majority (SPDZ)



Slowdowns grow, but *far* better than 1000x

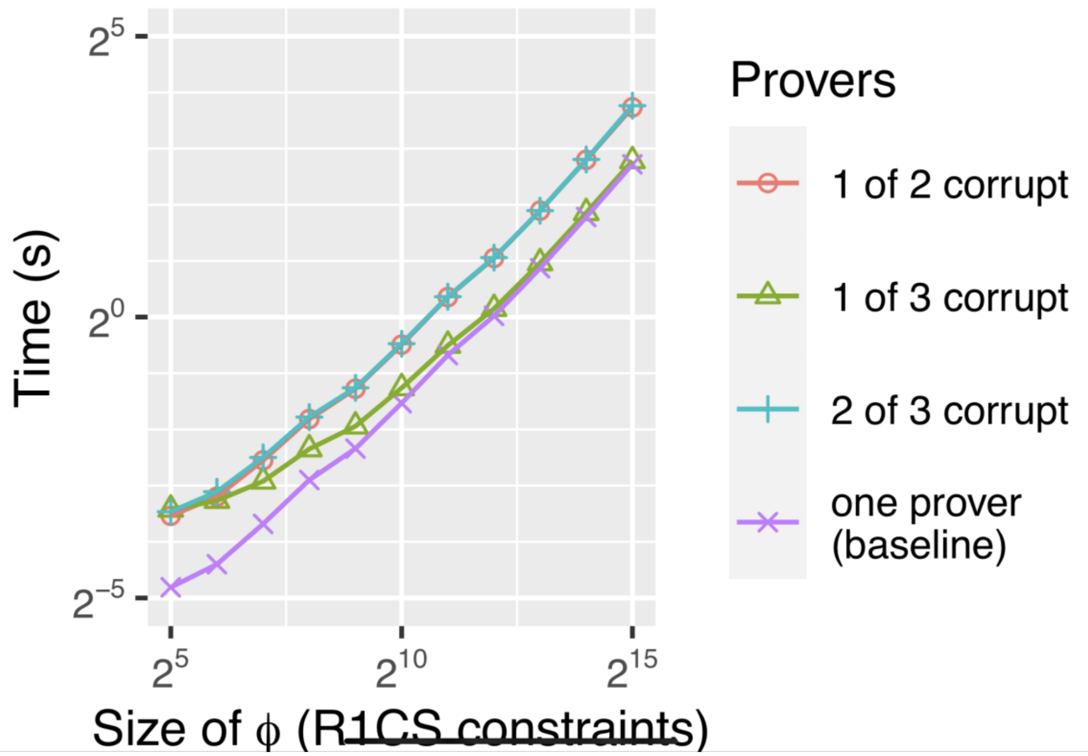
Discussion, Future Work

- Bandwidth is the bottleneck for many provers, low link capacity
  - Bad news: 2-prover co-zkSNARK (additive sharing) $\rightarrow$ $\Omega(n)$ communication
    - From randomized 2-party communication complexity of DISJOINT
    - Conjecture: $\Omega(\lambda n)$ (needed: generalize DISJOINT from $\{0,1\}$ to $\mathbb{F}$)
- Exploit intra-prover parallelism
- A nicer post-quantum co-zkSNARK with o(N) proof-size?

Collaborative zkSNARKs

Alex Ozdemir, Dan Boneh

Groth16 co-zkSNARK proving time



Conclusions:

1. **Collaborative zkSNARKs** support **distributed secrets**
   - Multiple users, hospitals, banks, …
2. Very efficient
   - $(N/2)$-ZK → **no slowdown**
   - $(N-1)$-ZK → **2x slowdown**
3. Far better than MPC for typical computations → **~1000x slowdown**

https://github.com/alex-ozdemir/multiprover-snark

https://ia.cr/2021/1530

Code

Component	Lines (Rust)
Network Library	~700
Arkworks adapters	~2000
MPC protocols	~3000
Plonk	~1200